International Journal of Grid and High Performance Computing

January-March 2010, Vol. 2, No. 1

Table of Contents

Research Articles

Modeling Scalable Grid Information Services with Colored Petri Nets

Vijay Sahota, Middlesex University, UK

Maozhen Li, Brunel University, UK

Marios Hadjinicolaou, Brunel University, UK

ABSTRACT

Information services play a crucial role in grid computing environments in that the state information of a grid system can be used to facilitate the discovery of resources and services available to meet user requirements and help tune the performance of the grid. This article models PIndex, which is a grouped peer-to-peer network with Colored Petri Nets (CPNs) for scalable grid information services. Based on the CPN model, a simulator is implemented for PIndex simulation and performance evaluation. The correctness of the simulator is further verified by comparing the results computed from the CPN model with the results generated by the PIndex simulator.

Keywords: Colored Petri Nets, Grid Computing, Information Services, P2P Modeling, P2P Networks

INTRODUCTION

The past few years have witnessed a rapid development of grid computing infrastructures and applications (Li & Baker, 2005; Wang, Helian, Wu, Deng, Khare, & Thompson, 2007; Wang, Wu, Helian, Parker, et al. 2007; Wang, Wu, Helian, Xu, 2007). Information services play a crucial role in grid environments in that they facilitate the discovery of resources and services (Czajkowski, Kesselman, Fitzgerald, & Foster, 2001). Information services periodically collect data on available resources including hardware and software in a grid environment. The data can then be used by a number of elements in a grid to keep the grid running smoothly. For example, job schedulers use resource information to make adaptive decisions on allocating resources to jobs to achieve certain goals such as a minimum make-span in execution of jobs (Berman et al., 2003).

Grid middleware technologies facilitate information services. For example, the current Globus Toolkit (http://www.globus.org) provides a component called the Monitoring and Discovery System version 4 (MDS4) (Schopf et al., 2006) for resource registration and discovery. The MDS4 component adopts

DOI: 10.4018/jghpc.2010092804

a hierarchical tree structure to distribute its monitoring data on resources across a virtual organization (VO), in which every node runs an index service monitoring its resources and pushing this information up to a master index server. A query to the top Index server could retrieve all the information on the resources available in a VO. The Relational Grid Monitoring Architecture (R-GMA) (Cooke et al., 2004), which is now a component of the gLite middleware (http://www.cern.ch/glite), also facilitates resource registration and discovery. It is worth noting that grids differentiate themselves from traditional distributed systems in the following aspects:

- The size of a grid is usually large in terms of the number of computing nodes involved.
- Resources in a grid are usually heterogeneous with various computing capabilities and services.
- A grid is dynamic in that computing nodes may join or leave a grid freely. In addition, some resources such as the CPU load of a grid node may change frequently.

The aforementioned characteristics of grids bring forth a number of challenges to existing information services, notably the MDS4 and the R-GMA. The hierarchical structure along with centralized management of MDS4 has an inherent delay associated with it which potentially limits its scalability in resource registration. It might take a long time for resource information to be updated from the leaf nodes to the root index service node. Cai, Frank, Chen, and Szekely (2004) point out that the scheme to partition resource information on index servers is typically predefined and cannot adapt to the dynamic changes of VOs. The MDS4 also lacks a mechanism to deal with failures of index servers which may break the information service network into isolated subnets. The R-GMA contains a centralized registry (Groep, Templon, & Loomis, 2006), and performs poorly when dealing with only 100 consumer nodes (Zhang, Freschl, & Schopf, 2007).

In parallel development with grid computing, peer-to-peer (P2P) computing has merged into another promising computing paradigm that typically facilitates file sharing in large network environments (Milojicic et al., 2002). P2P networks usually organize peer nodes in a decentralized way, and the reliability can be enhanced by replication of shared files among peer nodes. Files can be arbitrarily distributed into peer nodes without a structure, or they are distributed following a structure such as Distributed Hash Table (DHT). DHT based P2P networks such as Chord (Stoica et al., 2002), Pastry (Rowstron & Druschel, 2001), CAN (Ratnasamy, Francis, Handley, Karp, & Shenker, 2001) have shown enhanced scalability in routing lookup messages for files with a guaranteed number of hops. Foster and Iamnitchi (2003) analyzed the differences between P2P and grid computing and discussed a possible convergence of the two computing paradigms. Talia and Trunfio (2003) pointed out the benefits that P2P networks could bring to grid systems in terms of scalability and robustness. However, directly applying DHT technologies to grid information services mainly poses two challenges. On the one hand, DHT systems usually incur high maintenance overhead in dealing with *churn* situations where peer nodes may join or leave P2P networks at high rates (Godfrey, Shenker, & Stoica, 2006; Rhea, Geels, Roscoe, & Kubiatowicz, 2004). On the other hand, DHT based P2P networks only support exact matches for files using single hash keys. In a grid environment, it is not realistic to employ a single hash key for a resource which may have a number of attributes such as its CPU load, memory space, storage space, and availability. Moreover, the values of these resource attributes dynamically change in a grid environment. Range queries on resources should be supported in grid information services.

We have implemented PIndex (Sahota, Li, Baker, & Antonopoulos, 2009), which is

a grouped P2P network for scalable grid information services. PIndex has the following features:

- It builds on Globus MDS4, but introduces the concept of peer groups (PGs) to speed up the process in routing messages for resource discovery.
- It supports both static and dynamic routing of lookup messages for resources. In the latter case, a number of messages can be routed from one PG to another concurrently.
- It supports range queries searching for resources with a number of attributes.
- It enhances fault resilience through the replication of state information of resources among the computing nodes within a PG.
- It reduces the effects of *churn* by limiting them locally to a PG level and isolating the effects from the rest of the network.

PIndex is designed for large-scale grid systems. Simulations are required to observe its features and to evaluate its performance. It is worth noting that a number of grid simulators have been proposed. For example, SimGrid (Casanova, Legrand, & Quinson, 2008) aims at a generic evaluation tool for large-scale distributed computing. GridSim (Buyya & Murshed, 2002) was initially designed for grid economy but it only scales to a few 100 nodes as presented in (Casanova et al., 2008). OptorSim (Bell, Cameron, Capozza, Millar, Stockinger, & Zini, 2003) are specifically designed for data replications on grids. PlanetSim (López, Pairot, Mondéjar, Ahulló, Tejedor, & Rallo, 2004) and PeerSim (http://peersim.sourceforge.net) are proposed for the simulation of generic P2P applications. However, these simulators cannot be used for modeling and simulating PIndex because they cannot satisfy the following features of PIndex:

- Performing the specific message routing algorithm used by PIndex.
- Performing parallel PG operations.
- Modeling many thousands of nodes.

This article models PIndex with Colored Petri Nets (CPNs) (Jensen, Kristensen, & Wells, 2007). Based on the CPN model, a simulator is implemented for PIndex simulation and performance evaluation. Experimental results have shown that PIndex is scalable in routing messages among a large number of peer nodes up to 10,000 and is resilient in dealing with node failures (Sahota et al., 2009). The correctness of the PIndex simulator is also verified.

The remainder of the article is organized as follows. Next, we present the CPN modeling work on PIndex. We then implement a simulator based on the CPN model and verify the correctness of the PIndex simulator by comparing the results computed from the CPN model with the results generated by the simulator. We conclude the article by discussing future work for PIndex.

Modeling PIndex with Colored Petri Nets

A Colored Petri Net (CPN) is a modeling language that can be used to graphically represent the structure of a distributed system, employing places and transitions to model complex systems. More importantly, it supports concurrent processes. A Petri Net has place and transition nodes, as well as directed arcs connecting places with transitions. It is the tokens that transit on places representing a system moving from one state to another. The following main points have been made to further demonstrate the rationale in choosing CPNs to model PIndex:

- The nodes in PIndex may have many different states. These can be shown with colored tokens.
- As many PGs operating in parallel and interact with each other in PIndex, the ability to model concurrent events is needed. This requirement is intrinsic to Petri Nets as they are designed with concurrency in mind.

- Since PIndex is a robust and fault tolerant network, these features must to be able to be simulated. Petri Nets have the ability to change states during a transition, allowing for failures to occur randomly.
- Each node in PIndex must operate independently which can be mimicked by the use of tokens as objects in CPNs.

Modeling States

PIndex will encounter many states when placed in a real working environment. Before a CPN model can be established, firstly each state must be recognized. In our case a token represents a node on which its current state would be represented by its colour from which our CPN model will take its desired actions as shown in Table 1. These states coupled with its buffer, relative processing delay, bandwidth and node usage will be the parameters for every token. As it will be seen the requirement of buffers and an introduction of bandwidth consumption is introduced as additions to the current basic model, further more keeping a count of the number of nodes handling a request/state will also be factored in for logging purposes. A simple flow diagram will be used to represent the actions that are needed for each state thus their similarities can easily be seen, verifying the use of CPNs. However, critical to the accurate monitoring of PIndex's performance, resources such as time, bandwidth and node usage are used will be different for each state.

Modeling Search Requests

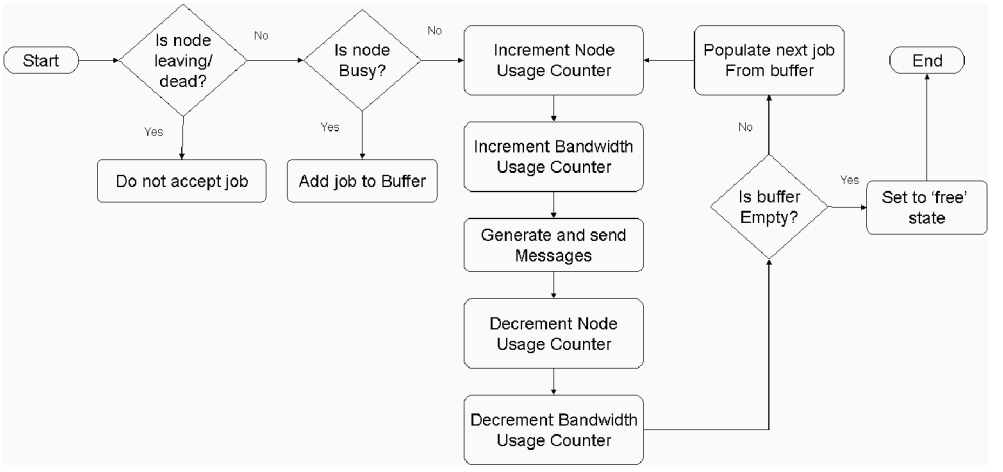
At the heart of PIndex is its ability to send a search and discover resources via its P2P algorithm. As shown in Figure 1, the first step is to check if the node is willing to accept a job (not dead or about to leave), if so it may carry on with the search request if not the message is lost, although in our model the assumption is made that the Table of External Contacts (TECs) are kept up-to-date without error, message loss is still possible in the case where a search arrives at a node that has just failed.

The next step would be to set the nodes current state to busy then increment a global counter of nodes being used for processing (total node usage), and a separate counter (PG node usage) is also kept for PG monitoring. Before the forwarded search messages are sent with a counter for bandwidth use incremented globally (total bandwidth) and locally (PG bandwidth), then the process of calculating the next nodes to query is executed and the message is sent, given its relative delay the next step is to release the

Table 1. The state table of PIndex

Colors	Description
Free	The node is free and able to process jobs.
Dead	The node is dead its position in the PG is vacant for joining nodes.
Busy	The node is busy processing a job, but will place a job in its buffer.
Leaving	The node will leave soon; it will not accept any more jobs but will complete the ones in its buffer. It will also tell its PG and referencing nodes to update their TEC's.
Search request	The node is sending a search request; the receiving node responds and forwards it according to the PIndex search algorithm.
Query Response	The node is sending a response back, to any form of query.
Join Requests	The node is requesting to join a PG (receiving), if space is vacant it will join, otherwise the request is forwarded to a neighbouring PG.
Contact Updates	The node is updating its TEC replacing its existing contacts with new ones.
Resources Updates	The node is updating its resource information.

Figure 1. The flow chart of search requests



bandwidth and node count being used. Before completing this state, a check is needed to see if the node's buffer is empty, if not a loop back is made to the processing stage, and if empty the node state is set to free.

Modeling Join Requests

In this situation a new node randomly selects a PG to join, on the condition there is a vacant space the node will be able to join, else the node will have to repeat its request in the next sequential PG. In this case we have a loop back condition using the discovery of a vacant space rather than checking if the buffer is empty.

Omitting the need to check if the node is alive or busy henceforth, the first step in Figure 2 is to take count of the node usage and bandwidth being used with its state set to *Busy*, then the actual request message is sent after which the resources are returned. The node receiving the request will then check if there are any vacant places in its PG, if not then the request is forwarded to its neighbouring PG. If there is a vacancy then it is given to the requesting node, its state and buffers are then set to free. The next logical step would be to invoke the *contact update* state to populate its TEC.

Modeling Leave Requests

When leaving, a node in PIndex must complete the tasks it already has in its buffer before leaving the network. Further tasks it may receive are not accepted in addition to telling its TEC contacts to update their references as it will be leaving soon. As shown in Figure 3, the flow starts by setting the node usage count, and consumes the required bandwidth resources and sends a message forcing its contacts to get new contact instead of itself whilst setting its state to *leaving*. After having executed all the tasks in its buffer the node is placed into a dead state

Modeling Resource Updates

The main concept of PIndex is to keep accurate and dynamic record of the resources available. This was achieved through the use of its update messages sent to all its local peers in its PG and any other node which has a reference to it. Figure 4 shows a resource update, the initial stage it consumes the resources required (node usage and bandwidth counter), and then proceeds in preparing a message to send to all (alive) contacts in it PG, and in its TEC. The message is then sent; note that if any nodes fail during this period no affect is had since update messages are sent in one way only. Given the

Figure 2. The flow chart of join requests

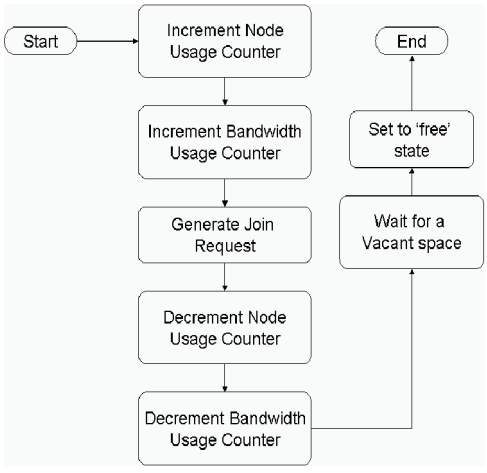
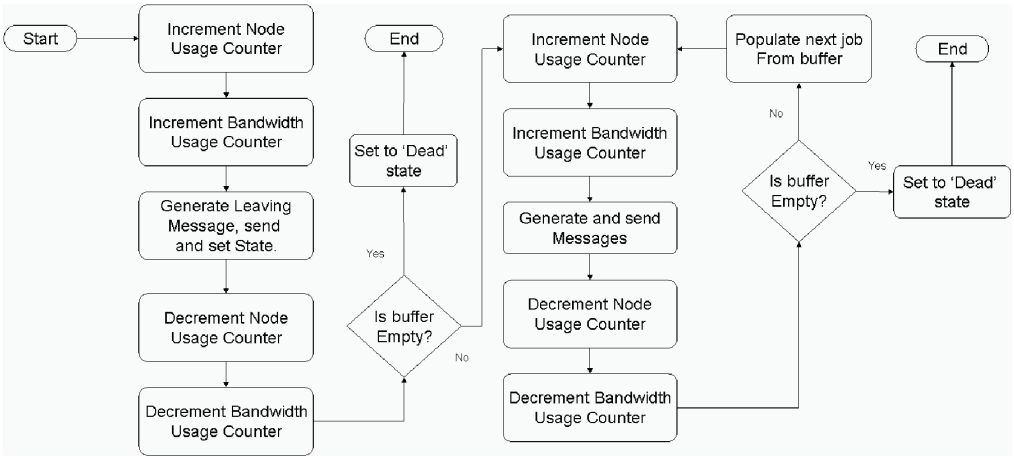


Figure 3. The flow chart of leave request



average time to send a message all the resources taken are then returned.

Modeling Query Responses

In this situation a node is responding to any request, and serves as a generic state any node can be in when responding to a request. For example, with a new TEC contact request, the node checks to see if it can forward and split the search space. Note: if the current node has some or all of the search criteria, it will also send a response back (query response).

The flow in Figure 5 starts by setting the node usage count, and consumes the required bandwidth resources; it then processes the request and responds accordingly after which the consumed resources are returned.

Figure 4. The flow chart of resource request

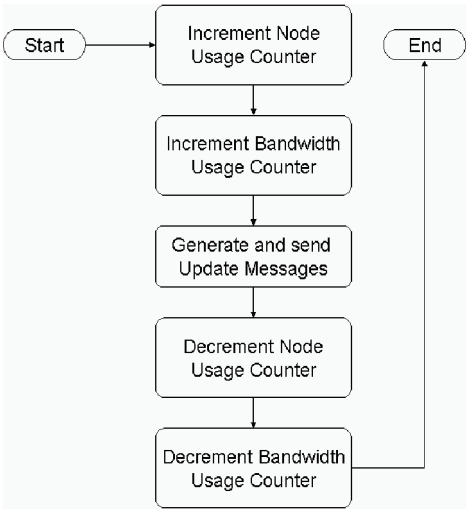
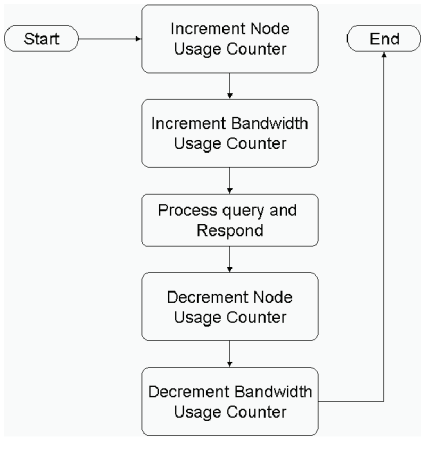


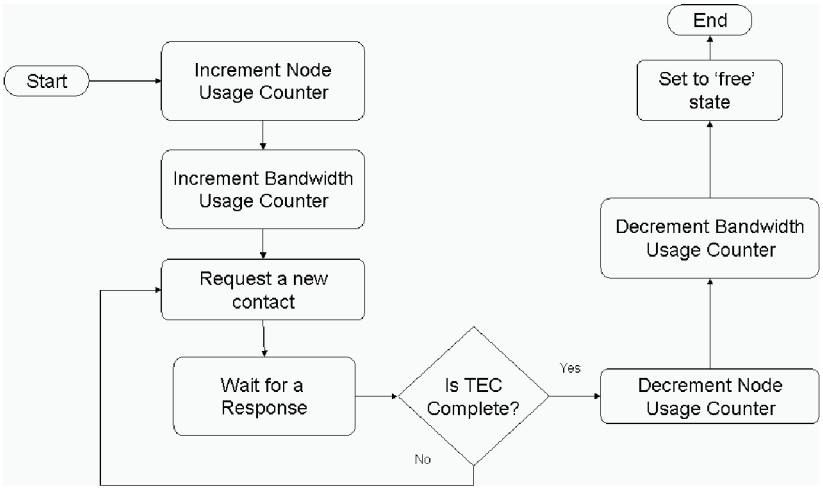
Figure 5. The flow chart of query responses



Modeling Contacts Updates

As part of the joining procedure (join state), newly joined nodes must update their TEC. This is a random process where the joining node randomly chooses a known external contact within the PG and queries them for a new contact in its PG for the joining node, this process is repeated to populate the new nodes TEC. This process is outlined in Figure 6, here the resources needed are consumed and then a request for a new contact is made (and response given), which is repeated to populate its TEC table. Note: not only does the new node gain an external contact, but the new contact must also add the new nodes ID to it list of nodes to

Figure 6. The flow chart of contact updates



send updates to. Once the table is populated, all consumed resources are returned and the node is set free.

The CPN Model

Figure 7 shows a generic flow chart of PIndex. There are 3 node states that do not consume resources—*Dead*, *Free* and *Busy*. These states are used as an indication of the current activity of a node. The remaining 6 node states are used to perform tasks consuming resources with some delays.

Figure 8 shows the CPN diagram of PIndex that has 9 places (P1-P9) and 6 transitions (t1-t6) which is explained in the following two cases:

Case 1: *A node is not busy and its buffer is empty. The token of the node is placed onto P1 firing t1 and goes to P2. Since the node is not busy, P3 will have no tokens and t2 will be fired. As t2 is fired it takes a token from both the node counter (P6) and message counter (P7) and places a token in P3 and P5 respectively. The amount of delay of the two delay transitions (t4 and t5) is dependent on the colors of the node. Since the buffer is empty, P4 has no tokens and t5 is fired which returns the token to both the node counter (P6) and message counter (P7). This*

Figure 7. The generic flow chart of PIndex

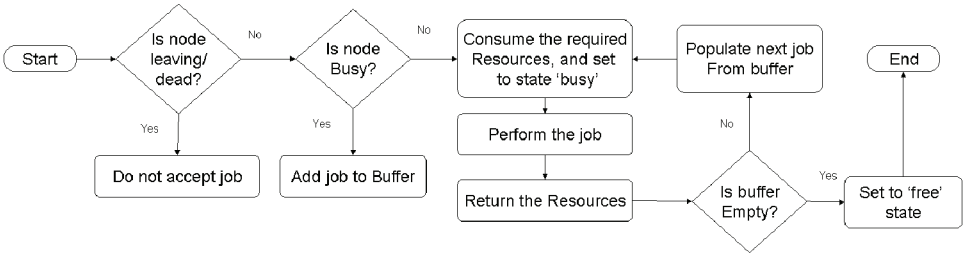
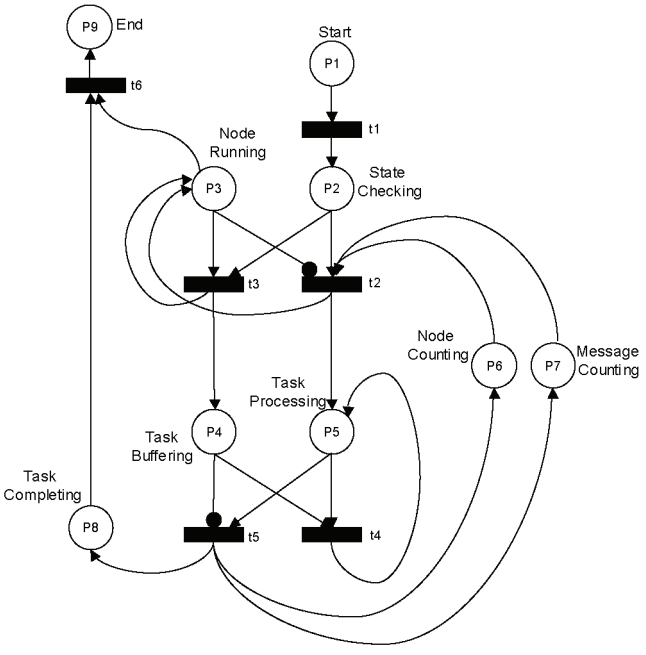


Figure 8. The CPN diagram of PIndex



will place a token onto P8. Since both P8 and P3 have a token, then t6 is fired and the token goes to P9 for completing the process.

Case 2: *A node is busy and its buffer is not empty. The token of the node is placed onto P1 firing t1 and goes to P2. Since the node is busy, P3 will have a token and t3 will be fired. As t3 is fired a token will be placed onto both P3 and P4. Since the buffer is not empty, t4 is fired which removes a token from both P4 and P5 and loops back to P5 to process the next task. This will keep looping until the buffer is empty which will then result in t5 firing returning the token to both the node counter (P6) and message counter (P7). This will place a token onto P8. Since both P8 and P3 have a token, then t6 is fired and the token goes to P9 for completing the process.*

PINDEX SIMULATOR

PIndex gains its speed in performance through PGs operating independently and concurrently. However although Petri Nets have been designed with concurrency in mind, running multiple instances that interact with each other is a not part of standard Petri Nets. Although a number of PNs could be combined into a single PN mimicking multiple instances with a derived mathematical model. However having a single PN would become cumbersome in that the number of PGs would be fixed, limiting the size of the network to be simulated, disabling the model in investigating PIndex's scalability. Moreover, creating a fixed model will limit its flexibility in investigating different network and routing conditions, such as forcing a certain type of PG grouping using a nodes probability of failure as a metric which would involve a full re-write of the CPN model and mathematics. In this section we take the independent PG nature of

PIndex, the CPN model and parallel executions of threads to produce an object orientated simulator. Using threads enables multiple instance of a PG to run concurrently, truly Modeling PIndex in a multi-network distribution (PIndex over many independent sites). The following section briefly describes the implementation of the PIndex simulator.

Simulator Implementation

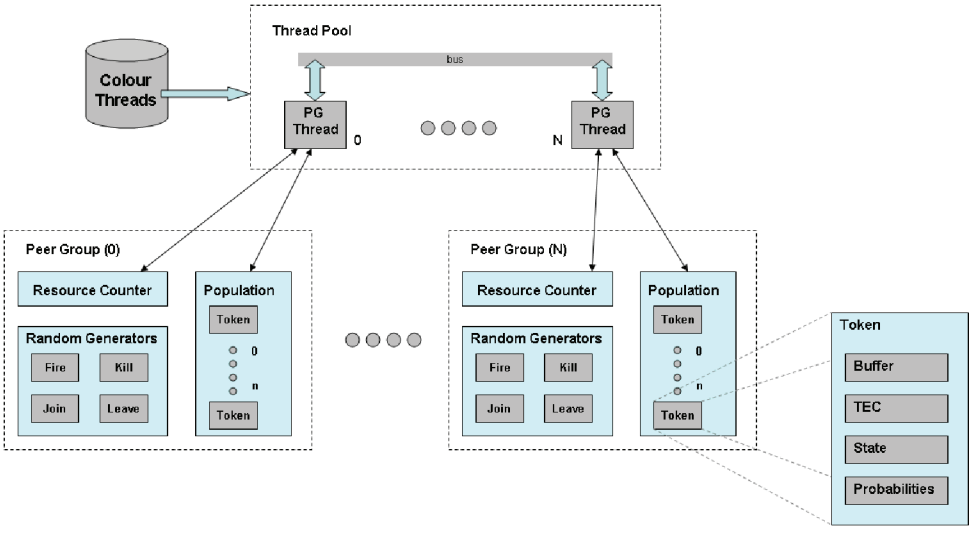
Based on CPN modeling, we have implemented a simulator using Java programming language for PIndex evaluation. Figure 9 shows the architecture of the PIndex simulator. Tokens are implemented as Java objects that contain a color state, a buffer array, a Table of External Contact (TEC), firing probabilities, a PG ID and a PG node ID. In order to model parallel executions of PGs in PIndex, each PG is implemented as a thread. Multiple threads are instantiated for the duration of the simulation creating a thread pool. Each PG has its own *Poisson* distribution generator, which provides *Poisson* rates for firing, updating, failures and leaving for the nodes in the PG. Each PG also has its own probabilistic normal distribution generator for its peer nodes. The probability of the peer nodes is used in such a way that when a peer node is chosen to run, a random percentage of probability is generated and compared with the probability of the node determining if the node runs.

Each colour state has an associated action as shown in Table 1, which was implemented as a thread. The reason for creating these coloured actions as threads is that, when a token is being processed in the PG, an instance of its respective colour action thread is created. These instances carry out the tasks that are specified (consume/return resources, delay and send/receive messages) and terminates once the job is done (while setting the token free/ execute next task in buffer). This frees the PG thread from executing the colour, before processing the next token in its queue allowing the current token to rejoin the PG (population) such that it can process further incoming request whilst its colour task is being executed.

When a peer node is chosen to run, the token object is passed into the PG thread which is queried. The PG thread checks if the token can accept the task:

- If the token is busy, then the task color is placed into the token's buffer.

Figure 9. The architecture of the PIndex simulator



- If the token is dead or leaving, then the task color cannot be accepted.
- If the token is free, then a thread will be created to process the task and some resources will be consumed accordingly.

Simulating Peer Nodes

To examine the performance of PIndex under heavy and dynamic conditions, our simulator must be capable of representing and processing many thousands of nodes, and exhibit dynamic characteristics such as choosing when to leave based on a nodes probability of leaving. In our solution nodes were represented as CPN tokens and so represented as objects in our simulator. Each token object has attributes consisting of the current state, a buffer, TEC, probabilities of firing, dying and leaving. Having tokens represented as objects not only simplified the process of creating tokens, but enabled the simulator to make as many instances of tokens (nodes), as was required to be simulated; each with its own independent buffer, states, TECs and probabilities. In addition, to their independent attributes using an object also meant that they could be passed to methods or threads in an exact way as tokens are passed from place to place (via transitions).

Simulating Peer Groups

PIndex is primarily based on the formation of PGs, such that their independent operations remove the need for a central store of information. In order to successfully implement PGs in our simulation they must exist as a single entity for the duration of the simulation, and possess the ability to process requests simultaneously. In our solution, PGs were represented as threads, with multiple instances forming a thread pool that lasted for the duration of the simulation. Since only a single thread is used to represent a PG, we must reduce the amount of processing per thread to a minimum so as not to create a bottleneck in the simulation. The PG threads are the actual implementation of the PIndex CPN, the only difference being is that the execution of a colour task is not handled by the PG thread, but by a created instance of a colour thread. This reduces the load of the PG thread, which means that a PG thread simply checks the state of the token and decides which thread to instantiate. An implementation in this way made it easier to simulate PIndex since the numbers of PG threads were fixed per simulation and easier to organize the priority level of tasks.

Simulating Resource Consumption

The main objective of PIndex is to produce an information service that can work efficiently over a dynamic network structure. Metrics for bandwidth and node usage must be measured during the simulation of PIndex, as these can be used to gauge the operational efficiency of PIndex. In order to make an accurate interpretation of PIndex a log of both metrics must be kept for the entire network and each individual PG. In our simulation a simple counter was kept for resource use, when a resource is being used an increment to the counter is made, and once completed (freeing the resource) the counter is decremented. A global counter was kept for resources regarding the entire network for each monitored resource, in addition to local counters. As these counters will vary in count as the simulation progressed, our simulator would record their values at the start of every simulated clock.

Simulating Discreet Time

To ensure proper execution order of tasks a simulated clock was needed. Having a discrete clock allowed our simulator to carry out multiple task before the clock rolled over, resulting in the executed task appearing to be carried out simultaneously in the simulation. In our simulation our discreet clock is implemented as a thread, which allowed the simulator to take advantage of thread priorities to ensure proper execution of each task (timing). The task priorities were as follows starting with the lowest.

- Simulated clock

- Colour state threads
- PG threads, highest being the pushing of jobs (firing)

Basically this order ensured that if nodes were to fail, this would be first to be carried out along with all the other firings. Then all the recently fired tokens would be processed, and then instantiated (colour threads) before finally incrementing the simulated clock.

VERIFYING THE PINDEX SIMULATOR

Before we used the CPN simulator to evaluate the performance of PIndex, we verified the correctness of the simulator based on CPN matrix calculations. In this section, we present the verification process.

Petri Net Matrices

In Petri nets (PNs), as tokens can enter a transition from a place (input) and leave a transition to a place (output), transitions can be represented in the form of matrices.

Let

- D^- represent the inputs of the transitions in a PN.
- D^+ represent the outputs of the transitions in a PN.
- D represent the transition matrix of a PN.

Therefore,

$$D = D^+ - D^- \tag{1}$$

Given the matrix D, an initial state matrix M, and an input vector E_j with all entries being equal to 0 except the j^{th} entry being equal to 1, then a PN can be modeled using equation (2) to calculate the next states of the PN which are represented by M'.

$$M' = M - E_j \cdot D^- + E_j \cdot D^+ = M + E_j \cdot D \tag{2}$$

Given that E_j only has a single entry equal to 1, this will cause the transition t_j to fire. If a sequence of input vectors is placed onto a $PN(E_i, E_j, E_k)$, then these will in turn set a sequence of transactions to fire (t_i, t_j, t_k) and produce the state matrix M' using equation (3).

$$M' = M + \left(E_i + E_j + E_k\right) \cdot D = M + X \cdot D \tag{3}$$

where X represents a firing vector.

Verifying the PIndex Simulator

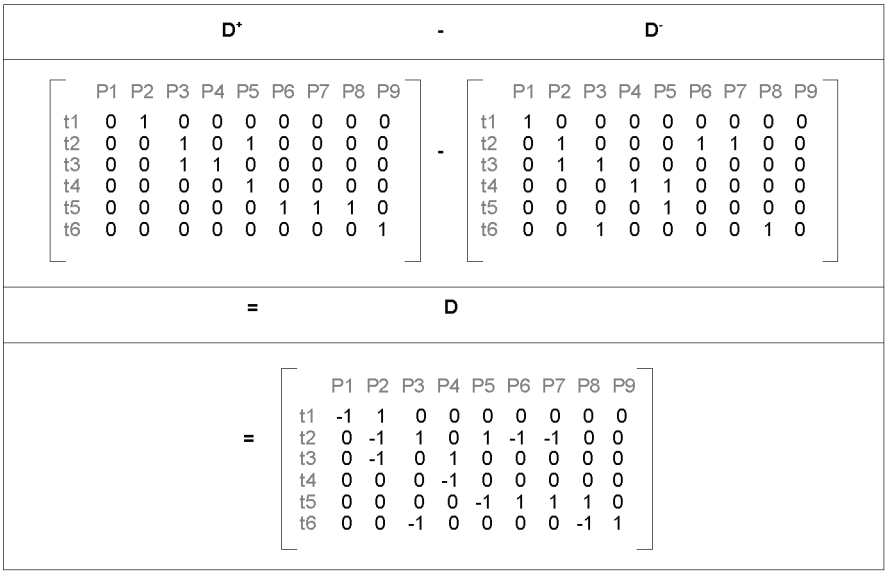
In this section, we verify the correctness of the PIndex simulator using CPN matrices. By applying colored states we filter out the correct color matrices to use, and are able to model the CPN of PIndex mathematically. Figure 10 shows the transition matrix D which is derived from the CPN diagram of PIndex shown in Figure 8.

We use a color vector C_i with a size of n to represent colors, where n is the total number of colors being used, and the i^{th} position represents the actual color. For example, if the color is for updating, then $C_4 = (0001)$. Having a transition matrix for each color, we need a way to isolate the transition matrix that receives the correct colored token. This can be achieved through the scalar product of the color vector of the current token (C_t) and the color vector of the associated transition matrix (C_d).

Given that

- C_t is the color vector of the current token.
- C_d is the color vector of the current transition matrix.
- $M_{transition}$ is the transition matrix of a color.

Figure 10. The D transition matrix of the CPN



- M is the previous state matrix.
- $M^{'}$ is the current state matrix.
- M_{color} is the resultant state matrix.

Then we have

$$M_{color} = \left(\left(Col\right)\cdot\left(\left(M^{'}\cdot C_{d}\right)\cdot M_{transition}\right)\right) + \left(\left(\overline{Col}\right)\cdot\left(\left(M\cdot C_{d}\right)\cdot M_{transition}\right)\right) \quad (4)$$

where $Col = C_{t}\cdot C_{d}$

Equation (4) is repeated for each transition matrix of every color, producing a resultant M_{color} in which each column represents the current number of tokens in each transition for that specific color. For the purpose of simplicity we use four colors in the CPN matrices to verify the correctness of the CPN simulator. Each color has the same transition matrix D. However, not all transitions are enabled because inhibitors are used at transitions t2 and t5 as shown in Figure 8. Therefore, an additional vector (σ) is used to represent which transitions are allowed to fire which is reflected in equation (5).

$$M_{color} = \left(\left(Col\right)\cdot\left(\left(M^{'}\cdot C_{d}\right)+M_{transition}\cdot\sigma\right)\right)+ \left(\left(\overline{Col}\right)\cdot\left(\left(M_{0}\cdot C_{d}\right)+M_{transition}\cdot\sigma\right)\right) \quad (5)$$

Given four color states as shown in Table 2, the test sequence used was 0001, 0100, 0100, 1000, 0010 and 0001.

Given M = 000009900 representing all places are empty, except the bandwidth and node buffer counters are both set to 9. Taking the first color vector (0001) of the sequence as an example, Table 3 shows the results of each of the $M_{trnansition}$ transition matrix and Table 4 shows the complete results of the sequence.

Figure 11 and Figure 12 show the theoretical results of the CPN transition matrix and the simulation results generated by the PIndex simulator from the aspects of bandwidth use and node usage respectively.

Table 2. The four colors used in the verification

Colors	Color Vectors
Resource Update	0001
Contact Update	0010
Search Request	0100
Search Response	1000

Table 3. The transition matrix results of the resource update

C_t	$M_{transition}$	Results	
0001	0001	Col	1
		σ	100000000
		M	000009900
		M_{color}	010009900
	0010	Col	0
		σ	000000000
		M	000009900
		M_{color}	000009900
	0100	Col	0
		σ	000000000
		M	000009900
		M_{color}	000009900
	1000	Col	0
		σ	000000000
		M	000009900
		M_{color}	000009900

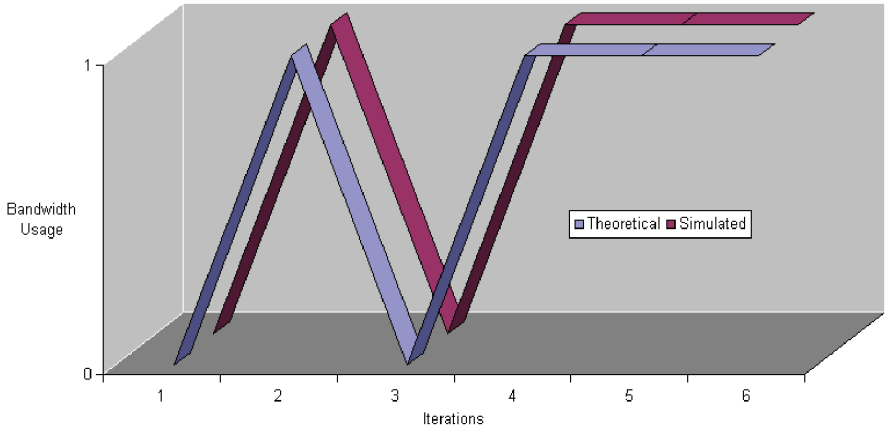
As shown in Figure 11, both the simulation and the theoretical results are the same in the aspect of bandwidth use. However there is a difference between simulation results and the theoretical results in terms of node usage as shown in Figure 12. We observe that in the 3rd iteration node usage remains 1 in the simulated results, whereas in the theoretical results node usage returns to 0. This variation in node usage is a direct result from the way the CPN simulator was implemented, in that as soon as a node receives a task its state is set to busy and does not change until the node is set free, that is, all tasks are completed and the buffer is empty. In the case of the theoretical results of the CPN, node usage changes as soon as a task starts or finishes which can be seen in the 3rd iteration where a task has completed and incremented the node counter whilst a state check was being carried out. This highlights how and where the CPN simulator has deviated from the CPN transition matrix which represents an ideal situation but is mathematically correct in reality.

Compared with the theoretical work of the CPN model, the PIndex simulator is more close to real grid information networks in which a peer

Table 4. The transition matrix results of the whole sequence

C_t	M_{color}
0001 (Resource Update)	010009900 000009900 000009900 000009900
0100 (Search Request)	001018800 000008800 010008800 000008800
0100 (Search Request)	001009910 011109900 001009900 001009900
1000 (Search Response)	000008801 000008800 001118800 010008800
0010 (Contact Update)	001008801 011008800 001018810 001108800
0001 (Resource Update)	011008801 001108800 001008811 001018800

Figure 11. Verification of bandwidth use

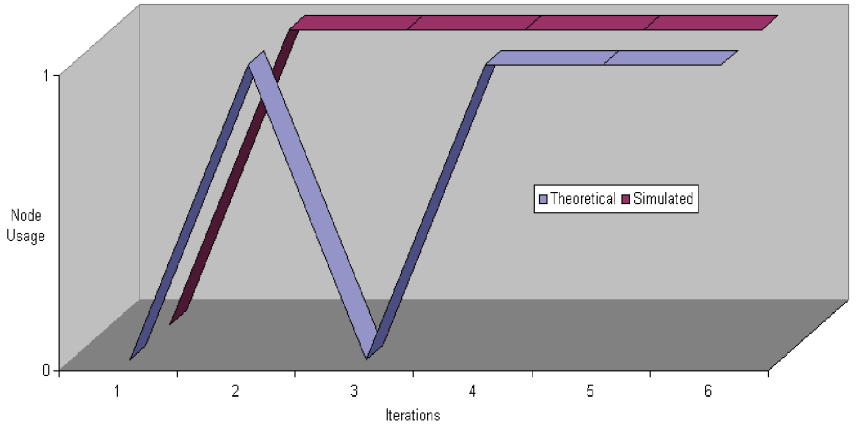


node is considered busy when it is running for various purposes including checking whether it is busy or not.

CONCLUSION

PIndex is a grouped P2P network which can be used as a substrate for grid information ser-

Figure 12. Verification of node usage



vices. This article modelled PIndex with CPN. Building on the CPN model, a PIndex simulator was implemented for PIndex simulation and performance evaluation. Experimental results have shown that PIndex is scalable when dealing with a large number of computing nodes up to 10,000, and is resilient in handling *churn* situations (Sahota et al., 2009). The correctness of the PIndex simulator was further verified. Future works include the implementation of the PIndex network and evaluate its performance in real grid environments.

REFERENCES

Bell, W. H., Cameron, D. G., Capozza, L., Millar, A. P., Stockinger, K., & Zini, F. (2003). OptorSim - a grid simulator for studying dynamic data replication strategies. *International Journal of High Performance Computing Applications*, *17*(4), 403–416. doi:10.1177/10943420030174005

Berman, F., Wolski, R., Casanova, H., Cirne, W., Dail, H., & Faerman, M. (2003). Adaptive computing on the grid using AppLeS. *IEEE Transactions on Parallel and Distributed Systems*, *14*(4), 369–382. doi:10.1109/TPDS.2003.1195409

Buyya, R., & Murshed, M. (2002). GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and Computation*, *14*(13-15), 1175–1220. doi:10.1002/cpe.710

Cai, M., Frank, M., Chen, J., & Szekely, P. (2004). MAAN: A multi-attribute addressable network for grid information services. *Journal of Grid Computing*, *2*(1), 3–14. doi:10.1007/s10723-004-1184-y

Casanova, H., Legrand, A., & Quinson, M. (2008, April 1-3). SimGrid: a generic framework for large-scale distributed experimentations. In *Proceedings of the 10th IEEE International Conference on Computer Modeling and Simulation (UKSIM/EUROSIM'08),* Cambridge, UK (pp. 126-131). Washington, D.C.: IEEE Computer Society.

Cooke, A. W., Gray, A. J. G., Nutt, W., Magowan, J., Oevers, M., & Taylor, P. (2004). The relational grid monitoring architecture: Mediating information about the grid. *Journal of Grid Computing*, *2*(4), 323–339. doi:10.1007/s10723-005-0151-6

Czajkowski, K., Kesselman, C., Fitzgerald, S., & Foster, I. (2001, August 7-9). Grid information services for distributed resource sharing. In *Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC 2001),* San Francisco (pp. 181-194). Washington, D.C.: IEEE Computer Society.

Foster, I., & Iamnitchi, A. (2003, February 20-21). On death, taxes, and the convergence of peer-to-peer and grid computing. In *Proceedings of the 2nd International Workshop on P2P Systems,* Berkeley, CA (pp. 118-128). University of California, Berkeley.

Godfrey, B. Shenker, S., & Stoica, I. (2006, September). Minimizing churn in distributed systems. In *Proceedings of ACM SIGCOMM Conference 2006*, Pisa, Italy (pp.147-158). ACM Publishing.

Groep, D. L., Templon, J., & Loomis, C. (2006). Crunching real data on the grid: Practice and experience with the European DataGrid. *Concurrency and Computation*, *18*(9), 925–940. doi:10.1002/cpe.962

Jensen, K., Kristensen, L. M., & Wells, L. (2007). Coloured petri nets and CPN Tools for Modeling and validation of concurrent systems. *International Journal on Software Tools for Technology Transfer*, *9*(3-4), 213–254. doi:10.1007/s10009-007-0038-x

Li, M., & Baker, M. (2005). *The grid: Core technologies*. London: John Wiley & Sons.

López, P., Pairot, C., Mondéjar, R., Ahulló, J., Tejedor, H., & Rallo, R. (2004, September). PlanetSim: A new overlay network simulation framework. In *Proceedings of the 4th International Workshop on Software Engineering and Middleware (SEM)*, Linz, Austria (pp. 123-136).

Milojicic, D. S., Kalogeraki, V., Lukose, R., Nagaraja, K., Pruyne, J., Rihard, B., et al. (2002). *Peer-to-peer computing* (Tech. Rep. HPL-2002-57). Palo Alto, CA: HP Labs.

Ratnasamy, S., Francis, P., Handley, M., Karp, R. M., & Shenker, S. (2001, August 27-31). A scalable content-addressable network. In *Proceedings of the ACM SIGCOMM Conference 2001,* San Diego, CA (pp.161-172). ACM Publishing.

Rhea, S., Geels, D., Roscoe, T., & Kubiatowicz, J. (2004, June 27-July 2). Handling churn in a DHT. In *Proceedings of the 2004 USENIX Annual Technical Conference,* Boston (pp. 127-140). USENIX.

Rowstron, A., & Druschel, P. (2001, November 12-16). Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware),* Heidelberg, Germany (pp. 329-350). ACM Publishing.

Sahota, V., Li, M., Baker, M., & Antonopoulos, N. (2009). A grouped P2P network for scalable grid information services. *Peer-to-Peer Networking and Applications*, *2*(1), 3–12. doi:10.1007/s12083-008-0016-4

Schopf, J. M., Pearlman, L., Miller, N., Kesselman, C., Foster, I., & D'Arcy, M. (2006). Monitoring the grid with the Globus Toolkit MDS4. *Journal of Physics: Conference Series*, *46*, 521–525. doi:10.1088/1742-6596/46/1/072

Stoica, I., Morris, R., Liben-Nowell, D., Karger, D. R., Kaashoek, M. F., & Dabek, F. (2002). Chord: A scalable peer-to-peer lookup protocol for Internet applications. *IEEE Transactions on Networks*, *11*(1), 17–32. doi:10.1109/TNET.2002.808407

Talia, D., & Trunfio, P. (2003). Toward a synergy between P2P and grids. *IEEE Internet Computing*, *7*(4), 94–96. doi:10.1109/MIC.2003.1215667

Wang, Z., Helian, N., Wu, S., Deng, Y., Khare, V., & Thompson, C. (2007). Grid-based storage architecture for accelerating bioinformatics computing. *VLSI Signal Processing*, *48*(3), 311–324. doi:10.1007/s11265-007-0066-5

Wang, Z., Wu, S., Helian, N., Parker, M., Guo, Y., & Deng, Y. (2007). Grid-oriented storage: A single-image, cross-domain, high-bandwidth architecture. *IEEE Transactions on Computers*, *56*(4), 474–487. doi:10.1109/TC.2007.1005

Wang, Z., Wu, S., Helian, N., Xu, Z., Deng, Y., & Khare, V. (2007). Grid-based data access to nucleotide sequence database. *New Generation Computing*, *25*(4), 409–424. doi:10.1007/s00354-007-0026-4

Vijay Sahota received his PhD from the School of Engineering and Design at Brunel University, UK in May 2008. He is now a post-doctoral research fellow at Middlesex University. His research interests are in the areas of distributed systems, grid computing specifically on grid information services, scalable peer-to-peer networks.

Maozhen Li is a lecturer in the School of Engineering and Design at Brunel University, United Kingdom. He received the PhD from Institute of Software, Chinese Academy of Sciences in 1997. He joined Brunel University as a full-time lecturer in 2002. His research interests are in the areas of grid computing, intelligent systems, P2P computing, semantic web, information retrieval, content based image retrieval. He has over 60 scientific publications in these areas. He authored The Grid: Core Technologies*, a well-recognized textbook on grid computing which was published by Wiley in 2005. He has served as an IPC member for over 30 IEEE conferences. He is on editorial boards of the* International Journal of Grid and High Performance Computing, the International Journal of Distributed Systems and Technologies, and the International Journal on Advances in Internet Technology.

Marios Hadjinicolaou received the BS degree in electronics from the University of London in 1979 and the MS and PhD degrees in electronic and electrical engineering from Brunel University, UK in 1982 and 1986 respectively. He is a senior lecturer in the School of Engineering and Design at Brunel University. His research interests are in the fields of Colored Petri Nets, video-on-demand, multimedia applications.